

## BÀI 7

# XỬ LÝ TRẠNG THÁI & LOGIC ỨNG DỤNG

**GV: ThS. BÙI PHÚ KHUYÊN**

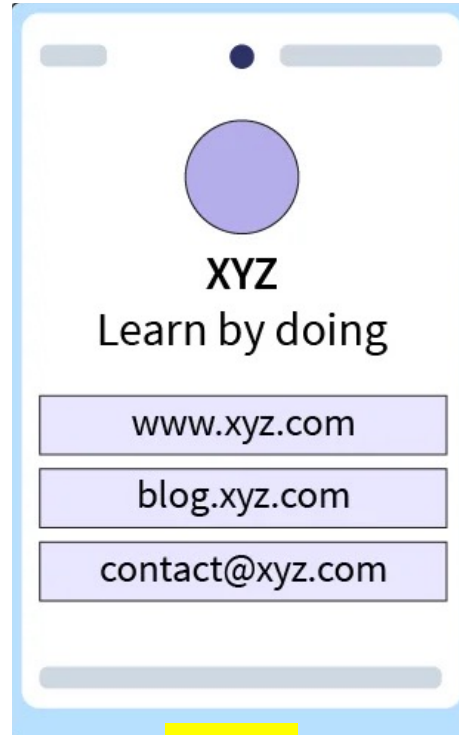
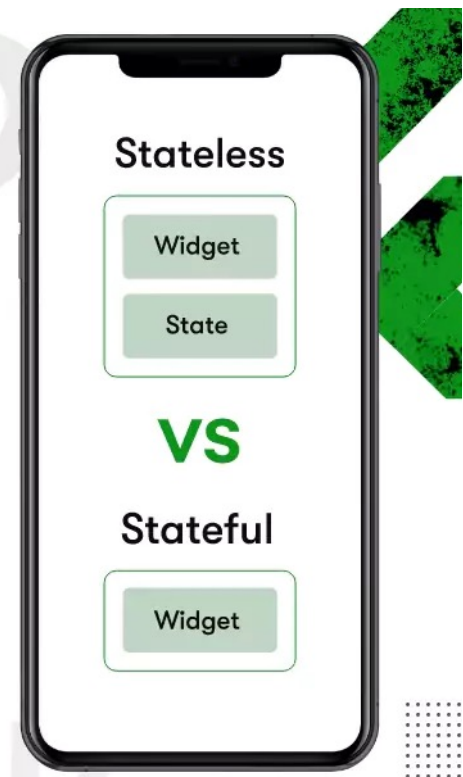


**\* Cập nhật: 15.10.2024**

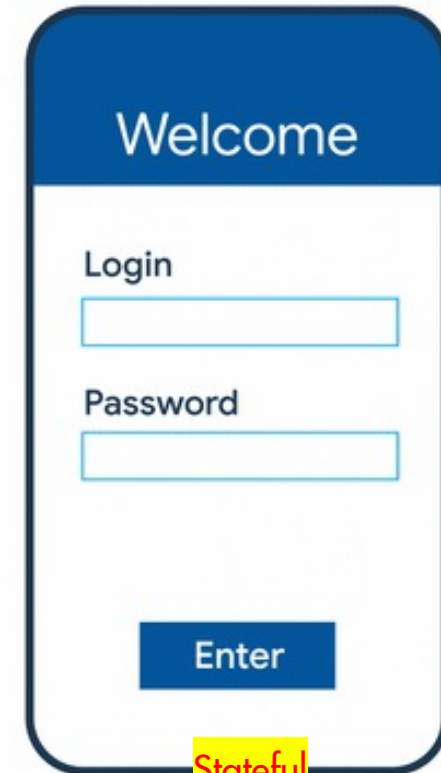
- ☐ Tìm hiểu kỹ về StatelessWidget và StatefulWidget
- ☐ Tầm quan trọng của quản lý trạng thái
- ☐ Tác động của việc lạm dụng setState()
- ☐ Giới thiệu về Provider
- ☐ Giới thiệu về BloC
- ☐ Giới thiệu về Get(X)



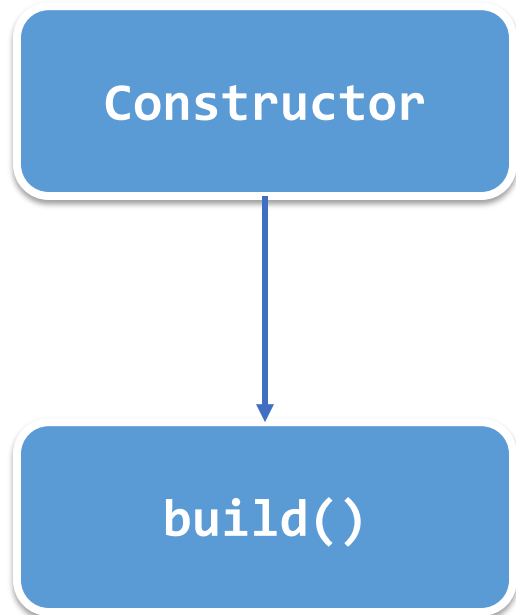
- **StatelessWidget**: Không có trạng thái, chỉ hiển thị dữ liệu cố định.
- **StatefulWidget**: Có thể thay đổi trạng thái và cập nhật giao diện khi cần.



Stateless



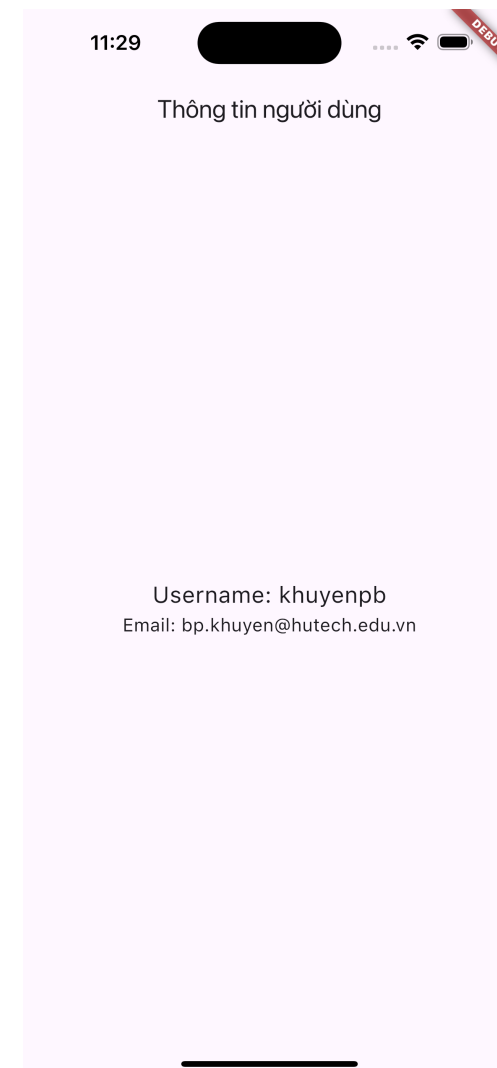
Stateful



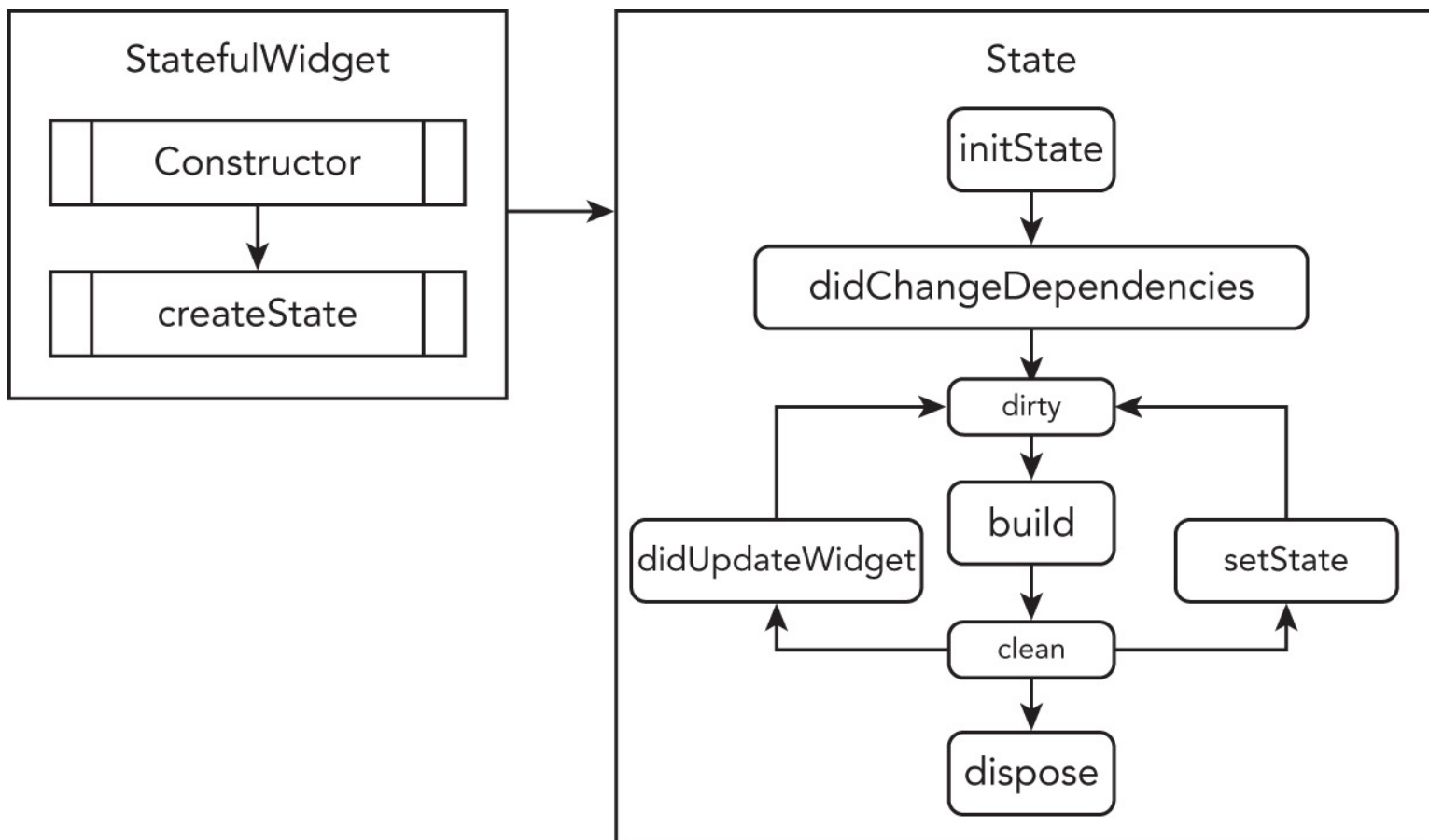
- **StatelessWidget** là một widget **không có trạng thái** bên trong. Nó chỉ dựa vào các tham số truyền vào qua constructor khi được tạo ra và **không thể thay đổi sau đó**.
- Phương thức **build()** được gọi **một lần** khi widget được đưa vào cây widget, và nó sẽ **không cập nhật lại giao diện** khi dữ liệu thay đổi.
- Phù hợp với **giao diện tĩnh** không cần cập nhật, chẳng hạn như văn bản, hình ảnh, hoặc các nút bấm cố định.

```
class UserInfoWidget extends StatelessWidget {  
  final String username;  
  final String email;  
  
  UserInfoWidget({required this.username, required this.email});  
  
  @override  
  Widget build(BuildContext context) {  
    return Column(  
      mainAxisAlignment: MainAxisAlignment.center,  
      children: <Widget>[  
        Text(data: 'Username: $username', style: const TextStyle(fontSize: 20)),  
        Text(data: 'Email: $email', style: const TextStyle(fontSize: 16)),  
      ],  
    ); // Column  
  }  
}
```

```
body: Center(  
  child: UserInfoWidget(  
    username: 'khuyenpb', email: 'bp.khuyen@hutech.edu.vn'),  
), // Center
```



## STATEFUL



**StatelessWidget** có thể thay đổi giao diện khi trạng thái bên trong thay đổi

```
class ScreenLifecycle extends StatefulWidget {  
  //Được gọi khi StatefulWidget được xây dựng  
  @override  
  _ScreenLifecycleState createState() => _ScreenLifecycleState();  
}  
  
class _ScreenLifecycleState extends State<ScreenLifecycle> {  
  //Khởi tạo các giá trị hoặc thiết lập các tài nguyên ban đầu  
  @override  
  void initState() {  
    super.initState();  
  }  
  //Xử lý thay đổi dependencies khi cần (chạy sau initState())  
  @override  
  void didChangeDependencies() {  
    super.didChangeDependencies();  
  }  
  //Xử lý khi widget cha thay đổi  
  @override  
  void didUpdateWidget(ScreenLifecycle oldWidget) {  
    super.didUpdateWidget(oldWidget);  
  }  
}
```

```
//Tạo giao diện (gọi lại khi có thay đổi trạng thái)  
@override  
Widget build(BuildContext context) {  
  return ElevatedButton(  
    onPressed: () {  
      //Gọi để cập nhật giao diện  
      setState(() {  
        print('setStat');  
      });  
    },  
    child: const Text('Press Me'),  
  );  
}  
//Xử lý và giải phóng tài nguyên khi widget bị xóa  
@override  
void deactivate() {  
  super.deactivate();  
}  
  
@override  
void dispose() {  
  super.dispose();  
}  
}
```



- **Local State Management:** Quản lý trạng thái cục bộ trong widget. (Sử dụng `setState()`)
- **Global State Management:** Quản lý trạng thái toàn cục, chia sẻ giữa nhiều widget hoặc toàn bộ ứng dụng. (Sử dụng các thư viện: Provider, Get, Bloc)

```
int _counter = 0;
```

```
void _incrementCounter() {  
  setState(() {  
    _counter++;  
  });  
}
```

```
floatingActionButton: FloatingActionButton(  
  onPressed: _incrementCounter,  
  tooltip: 'Increment',  
  child: Icon(Icons.add),  
),
```

Ví dụ minh họa sử dụng `setState()` để quản lý trạng thái cục bộ



- Phức tạp và khó bảo trì
- Khả năng mở rộng kém
- Hiệu năng kém
- Khó khăn trong việc chia sẻ trạng thái

```
int _counter = 0;
List<String> _items =
List.generate(100, (index) => '$index');

void _incrementCounter() {
  setState(() {
    _counter++;
  });
}

floatingActionButton: FloatingActionButton(
  onPressed: _incrementCounter,
),
```

@override

```
Widget build(BuildContext context) {
  return Column(
    children: [
      // Hiển thị giá trị bộ đếm
      Text('Counter: $_counter'),
      Expanded(
        // Danh sách các mục
        child: ListView.builder(
          itemCount: _items.length,
          itemBuilder: (context, index) {
            return ListTile(
              title: Text(_items[index]),
            );
          },
        ),
      ),
    ],
  );
}
```



# Battle of the State Managers:

**BLOC**

VS

**PROVIDER**

VS

**GETX**

# TÌM HIỂU CƠ BẢN

# PROVIDER LIBRARY

**Provider** là một thư viện quản lý trạng thái phổ biến trong Flutter, cung cấp cách thức đơn giản và hiệu quả để chia sẻ và quản lý trạng thái giữa các widget.

- **Dễ sử dụng:** Cấu trúc đơn giản, dễ hiểu, rất phù hợp cho những người mới bắt đầu quản lý trạng thái toàn cục.
- **Hiệu suất cao:** Provider chỉ tái tạo lại các widget thực sự cần cập nhật, giúp tối ưu hiệu suất.
- **Khả năng mở rộng:** Provider dễ dàng kết hợp với các mẫu kiến trúc phức tạp khác như MVVM hoặc BloC.
- **Khuyến nghị bởi Google:** Provider được khuyến khích bởi Google cho các ứng dụng Flutter, nhờ tính đơn giản và hiệu quả của nó.

provider 6.1.2

Published 7 months ago • [dash-overflow.net](#) Dart 3 compatible

[SDK](#) [FLUTTER](#) [PLATFORM](#) [ANDROID](#) [IOS](#) [LINUX](#) [MACOS](#) [WEB](#) [WINDOWS](#)

[Readme](#) [Changelog](#) [Example](#) [Installing](#) [Versions](#) [Scores](#)

[English](#) | [French](#) | [Português](#) | [简体中文](#) | [Español](#) | [한국어](#) | [বাংলা](#) | [日本語](#) | [Turkish](#)

[Build](#) [passing](#) [codecov](#) [99%](#) [chat](#) [260 online](#)



## Cài đặt thư viện

### provider 6.1.2

Published 7 months ago • [dash-overflow.net](#) Dart 3 compatible

[SDK](#) [FLUTTER](#) [PLATFORM](#) [ANDROID](#) [IOS](#) [LINUX](#) [MACOS](#) [WEB](#) [WINDOWS](#)

 10.2K

[Readme](#) [Changelog](#) [Example](#) [Installing](#) [Versions](#) [Scores](#)

### Use this package as a library

Depend on it

Run this command:


With Flutter:

```
$ flutter pub add provider
```



This will add a line like this to your package's pubspec.yaml (and run an implicit `flutter pub get`):

```
dependencies:  
  provider: ^6.1.2
```



## Tạo lớp trạng thái (State Class)

Tạo một lớp trạng thái sử dụng **ChangeNotifier**. Lớp này sẽ chứa trạng thái cần chia sẻ và các phương thức cập nhật trạng thái.

```
/// Lớp Counter kế thừa từ ChangeNotifier, cho phép thông báo các widget lắng  
nghe khi có sự thay đổi trong trạng thái.  
class Counter extends ChangeNotifier {  
    int _count = 0;           //Trạng thái ban đầu của bộ đếm _count  
  
    int get count => _count;   //Phương thức getter để truy cập biến _count  
  
    //Phương thức tăng giá trị cho _count  
    void increment() {  
        _count++;  
        notifyListeners(); // Thông báo cập nhật cho các widget lắng nghe  
    }  
}
```

Ví dụ về lớp **Counter** để quản lý số đếm.

## Cung cấp trạng thái với ChangeNotifierProvider trong Widget gốc (main)

Sử dụng ChangeNotifierProvider để cung cấp trạng thái Counter cho các widget con.

```
void main() {  
  runApp(  
    //ChangeNotifierProvider cung cấp trạng thái Counter cho toàn bộ ứng dụng  
    ChangeNotifierProvider(  
      create: (context) => Counter(), // Khởi tạo trạng thái Counter  
      child: MyApp(),  
    ),  
  );  
}
```

Sử dụng một ChangeNotifierProvider

```
void main() {  
  runApp(  
    MultiProvider(  
      providers: [  
        ChangeNotifierProvider(create: (context) => Counter()),  
        ChangeNotifierProvider(create: (context) => AnotherModel()),  
      ],  
      child: MyApp(),  
    ),  
  );  
}
```

Sử dụng nhiều ChangeNotifierProvider

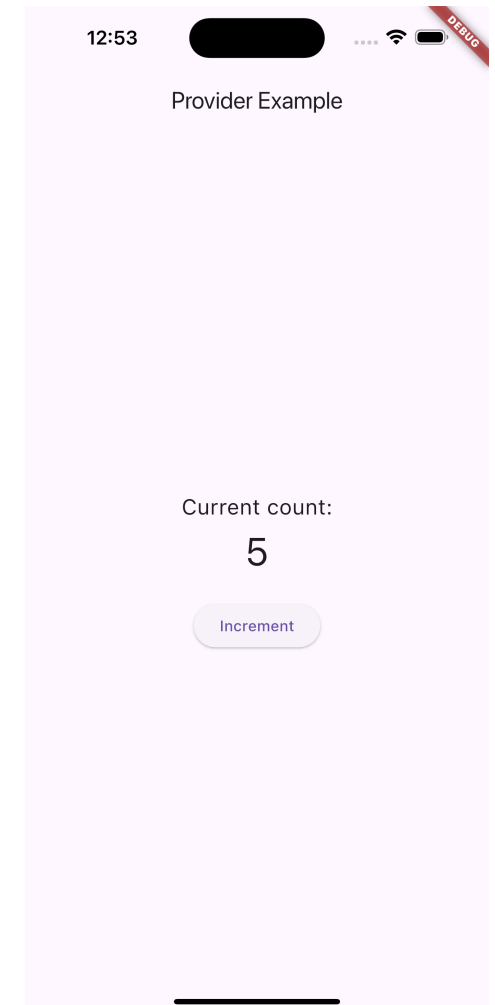


## Truy cập trạng thái bằng cách sử dụng Consumer hoặc Provider.of

- **Provider.of()**: Khi muốn truy cập trực tiếp đối tượng (không dùng **builder**).  
Và có thể không cần lắng nghe và cập nhật lại giao diện (**listen: false**)
- **Consumer()**: Luôn lắng nghe sự thay đổi: Bất kỳ khi nào đối tượng Counter thay đổi, widget sử dụng Consumer sẽ tự động tái tạo lại. (cần dùng **builder**)

```
// Truy cập vào Counter và gọi increment nhưng không lắng nghe thay đổi
Provider.of<Counter>(context, listen: false).increment();
```

```
// Lắng nghe và xây dựng lại giao diện khi Counter thay đổi
Consumer<Counter>(
  builder: (context, counter, child) {
    return Text('Count: ${counter.count}');
  },
);
```



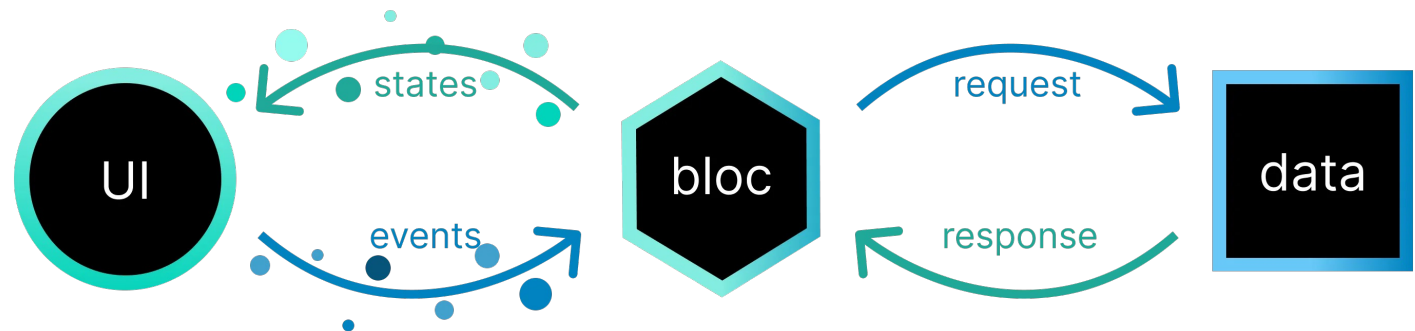
# TÌM HIỂU CƠ BẢN

# BLOC LIBRARY

**BloC** (Business Logic Component) là một mô hình quản lý trạng thái trong Flutter giúp tách biệt rõ ràng giữa giao diện người dùng (presentation) và logic xử lý nghiệp vụ (business logic).

## Các lớp trong mô hình BloC

- Data Layer
  - Data Provider
  - Repository
- Business Logic Layer (BLoC Layer)
- Presentation Layer



## Cài đặt thư viện

**flutter\_bloc** Tích hợp logic với giao diện người dùng trong Flutter

7187 160 100%  
LIKES PUB POINTS POPULARITY

Flutter Widgets that make it easy to implement the BLoC (Business Logic Component) design pattern. Built to be used with the bloc state management package. [#bloc](#) [#state-management](#)

v 8.1.6 (4 months ago) [bloclibrary.dev](#) MIT [Flutter Favorite](#) [Dart 3 compatible](#)

SDK FLUTTER PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS

API results: ▶ [flutter\\_bloc/Bloc-class.html](#)



**bloc** Tạo và quản lý logic trạng thái

2870 160 99%  
LIKES PUB POINTS POPULARITY

A predictable state management library that helps implement the BLoC (Business Logic Component) design pattern. [#bloc](#) [#state-management](#)

v 8.1.4 (6 months ago) [bloclibrary.dev](#) MIT [Flutter Favorite](#) [Dart 3 compatible](#)

SDK DART FLUTTER PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS

API results: ▶ [bloc/bloc-library.html](#)



## Khởi tạo Observer với BlocObserver

BlocObserver cho phép quan sát các thay đổi trạng thái toàn cục trong ứng dụng.

```
import 'package:bloc/bloc.dart';

/// Tạo lớp kế thừa từ BlocObserver
/// để theo dõi các thay đổi trạng thái toàn cục trong ứng dụng.
class CounterObserver extends BlocObserver {
  ///Phương thức được gọi khi có sự thay đổi trạng thái trong Bloc hoặc Cubit
  @override
  void onChange(BlocBase<dynamic> bloc, Change<dynamic> change) {
    super.onChange(bloc, change);
    // Ghi log loại bloc và trạng thái mới.
    print('${bloc.runtimeType} $change');
  }
}
```

## Tầng Logic - Định nghĩa CounterCubit

**Cubit** là một loại lớp logic để quản lý trạng thái mà không cần sự kiện, chỉ đơn giản là gọi các phương thức.

```
import 'package:bloc/bloc.dart';

//CounterCubit kế thừa từ Cubit<int> để quản lý trạng thái của một bộ đếm
class CounterCubit extends Cubit<int> {
  // Khởi tạo với trạng thái là 0.
  CounterCubit() : super(0);
  /// Phương thức tăng giá trị bộ đếm.
  void increment() => emit(state + 1);
  /// Phương thức giảm giá trị bộ đếm.
  void decrement() => emit(state - 1);

  //Chú thích: Phương thức emit() phát ra trạng thái mới.
}
```

Lưu ý: Sinh viên có thể tìm hiểu thêm về lớp logic Bloc trong Giáo trình (giống Cubit)

## Tầng Giao diện - Xây dựng CounterView với BlocBuilder

**BlocBuilder:** Một widget lắng nghe các thay đổi trạng thái từ **Cubit** hoặc **Bloc** và xây dựng lại giao diện khi trạng thái thay đổi.

```
class CounterView extends StatelessWidget {  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      body: Center(  
        //BlocBuilder lắng nghe thay đổi của CounterCubit và  
        //xây dựng lại giao diện khi trạng thái thay đổi.  
        child: BlocBuilder<CounterCubit, int>(  
          builder: (context, state) {  
            return Text('$state');  
          },  
        ),  
      ),  
    ),  
  ),  
);  
  
//Lấy instance của CounterCubit để gọi các phương thức  
//tăng/giảm giá trị  
floatingActionButton: Column(  
  mainAxisAlignment: MainAxisAlignment.end,  
  children: [  
    FloatingActionButton(  
      onPressed: () =>  
        context.read<CounterCubit>().increment(),  
      child: const Icon(Icons.add),  
    ),  
    FloatingActionButton(  
      onPressed: () =>  
        context.read<CounterCubit>().decrement(),  
      child: const Icon(Icons.remove),  
    ),  
  ],  
),
```



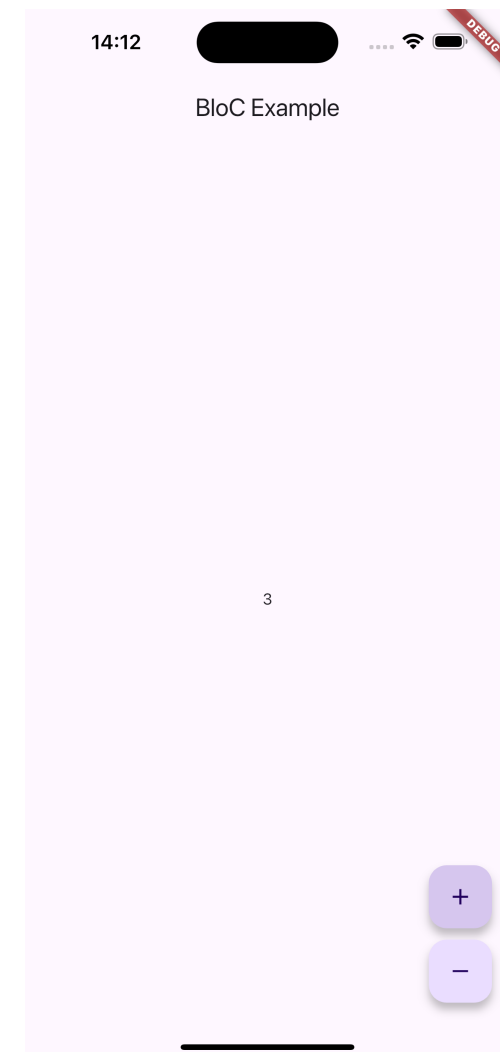
## Tầng Giao diện - Cung cấp CounterCubit với BlocProvider

**BlocProvider:** Một widget cung cấp Cubit hoặc Bloc cho các widget con trong cây widget.

```
class CounterPage extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    //BlocProvider tạo một instance của CounterCubit  
    //và cung cấp nó cho các widget con  
    return BlocProvider(  
      create: (_) => CounterCubit(),  
      child: const CounterView(), //Sử dụng CounterView là widget con.  
    );  
  }  
}
```

## Định nghĩa Bloc trong main

```
void main() {  
  ///Đăng ký Observer để theo dõi mọi thay đổi trạng thái trong ứng dụng  
  Bloc.observer = CounterObserver();  
  runApp(  
    MyApp(),  
  );  
}  
  
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      home: Scaffold(  
        appBar: AppBar(title: const Text('Bloc Example')),  
        body: const CounterPage(),  
      );  
    }  
}
```



# TÌM HIỂU CƠ BẢN

# GET(X) LIBRARY

## GetX là gì?

- Thư viện hiệu suất cao cho Flutter.
- Cung cấp quản lý trạng thái, quản lý phụ thuộc (DI) và khả năng điều hướng route mạnh mẽ.
- Tối ưu hoá hiệu suất và giảm thiểu tiêu thụ tài nguyên.

## 3 nguyên tắc chính của GetX

- Hiệu suất: Tối ưu hóa cho hiệu suất và tiêu thụ ít tài nguyên.
- Năng suất: Syntax đơn giản, dễ sử dụng, giúp tăng tốc độ phát triển.
- Tổ chức: Hỗ trợ tách biệt View, Model, Controller



## Cài đặt thư viện

get 4.6.6 

Published 13 months ago • [getx.site](https://pub.dev/packages/get) Dart 3 compatible • Latest: 4.6.6 / Prerelease: 5.0.0-release-candidate-9.2.1

[SDK](#) [FLUTTER](#) [PLATFORM](#) [ANDROID](#) [IOS](#) [LINUX](#) [MACOS](#) [WEB](#) [WINDOWS](#)

 14.7K

[Readme](#) [Changelog](#) [Example](#) [Installing](#) [Versions](#) [Scores](#)

### Use this package as a library

Depend on it

Run this command:

With Flutter:

```
$ flutter pub add get
```

This will add a line like this to your package's pubspec.yaml (and run an implicit `flutter pub get`):

```
dependencies:  
  get: ^4.6.6
```

## Tạo GetController cho Counter

**CounterController** sẽ quản lý trạng thái của ứng dụng và chứa các phương thức để tăng và giảm giá trị đếm.

```
import 'package:get/get.dart';

//GetxController Được tạo để lưu trữ trạng thái và các logic liên quan đến ứng dụng.
//Các biến với .obs là các biến quan sát được (có trạng thái).
class CounterController extends GetxController {
  var count = 0.obs; // Khởi tạo biến quan sát được với giá trị ban đầu là 0

  void increment() => count++; // Hàm tăng giá trị count
  void decrement() => count--; // Hàm giảm giá trị count
}
```

## Tạo UI (View) cho CounterApp

Sử dụng **Obx** để tự động cập nhật UI khi count thay đổi. Sử dụng **Get.put()** để khởi tạo và cung cấp CounterController.

```
class CounterApp extends StatelessWidget {
  //Khởi tạo CounterController qua cơ chế D.I
  final CounterController c = Get.put(CounterController());

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text("GetX Counter App")),
      body: Center(
        // Sử dụng Obx để lắng nghe thay đổi của count
        // và tự động cập nhật Text
        child: Obx(() =>
          Text("Count: ${c.count}"),
        ),
      ),
    );
  }
}
```

```
floatingActionButton: Column(
  mainAxisAlignment: MainAxisAlignment.end,
  children: [
    FloatingActionButton(
      onPressed: () => c.increment(),
      child: const Icon(Icons.add),
    ),
    const SizedBox(height: 10),
    FloatingActionButton(
      onPressed: () => c.decrement(),
      child: const Icon(Icons.remove),
    ),
  ],
),
...

```



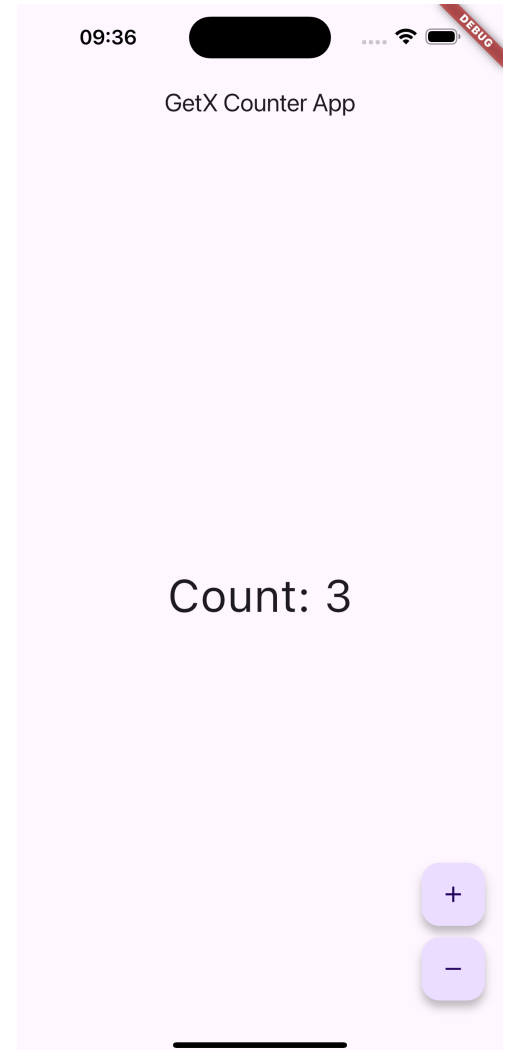
## Cấu hình main.dart

Sử dụng `GetMaterialApp` thay vì `MaterialApp` để hỗ trợ các tính năng của `GetX` như điều hướng và quản lý trạng thái.

```
void main() {  
  runApp(MaterialApp(  
    home: CounterApp(),  
  ));  
}
```

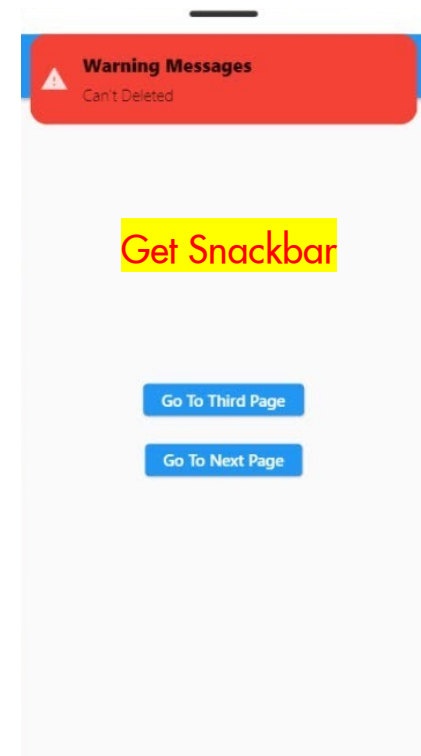
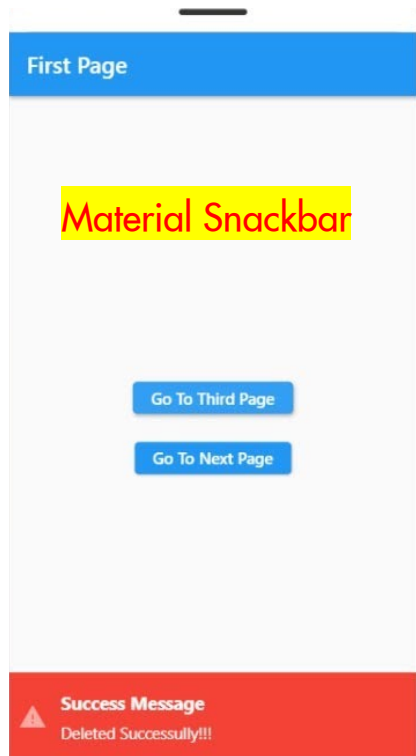


```
void main() {  
  runApp(GetMaterialApp(  
    home: CounterApp(),  
  ));  
}
```



## SNACKBAR

Đễ dàng hiển thị Snackbar  
mà không cần truyền context



```
ScaffoldMessenger.of(context).showSnackBar(
  SnackBar(
    content: Text('Hello! This is a Snackbar.'),
  ),
);
```

```
Get.snackbar('Hi', 'This is a GetX Snackbar');
```

## DIALOG...BOTTOMSHEETS...

```
showDialog(
  context: context,
  builder: (context) {
    return AlertDialog(
      title: Text('Hello'),
      content: Text('This is a Material Dialog'),
      actions: <Widget>[
        TextButton(
          onPressed: () => Navigator.of(context).pop(),
          child: Text('OK'),
        ),
      ],
    );
  },
);
```

Material Dialog

```
Get.defaultDialog(
  title: 'Hello',
  content: Text('This is a GetX Dialog'),
);
```

Get Dialog

## ĐIỀU HƯỚNG {1}

- Điều hướng sang màn hình mới (Push)

```
Navigator.of(context).push(  
  MaterialPageRoute(  
    builder: (context) => NextScreen()  
  ),  
);
```

```
Get.to(NextScreen());
```

- Điều hướng và loại bỏ màn hình hiện tại (PushReplacement)

```
Navigator.of(context).pushReplacement(  
  MaterialPageRoute(  
    builder: (context) => NextScreen()  
  ),  
);
```

```
Get.off(NextScreen());
```

## ĐIỀU HƯỚNG {2}

- Điều hướng và loại bỏ tất cả các màn hình trước đó (PushAndRemoveUntil)

```
Navigator.of(context).pushAndRemoveUntil(  
  MaterialPageRoute(  
    builder: (context) => NextScreen()),  
  (Route<dynamic> route) => false,  
);
```

```
Get.offAll(NextScreen());
```

- Quay lại màn hình trước đó (Pop)

```
Navigator.of(context).pop();
```

```
Get.back();
```

## ĐIỀU HƯỚNG {3}

- Điều hướng với tên đường dẫn (Named Route)

```
Navigator.of(context).pushNamed('/nextScreen');
```

```
Get.toNamed('/nextScreen');
```

- Điều hướng với tên đường dẫn và loại bỏ màn hình hiện tại (Named Route Replacement)

```
Navigator.of(context).pushReplacementNamed('/nextScreen');
```

```
Get.offNamed('/nextScreen');
```

- Điều hướng với tên đường dẫn và loại bỏ tất cả các màn hình trước đó (Named Route with Remove Until)

```
Navigator.of(context).pushNamedAndRemoveUntil(  
  '/nextScreen',  
  (Route<dynamic> route) => false,  
);
```

```
Get.offAllNamed('/nextScreen');
```

## QUẢN LÝ KEY-VALUE DỄ DÀNG VỚI GETSTORAGE

GetX cung cấp **GetStorage** để lưu trữ dữ liệu key-value mà không cần khởi tạo như SharedPreferences.

```
SharedPreferences prefs = await  
SharedPreferences.getInstance();  
await prefs.setString('key', 'value');  
String value = prefs.getString('key');
```

```
final box = GetStorage();  
box.write('key', 'value');  
String value = box.read('key');
```



## ĐỔI NGÔN NGỮ (INTERNATIONALIZATION)

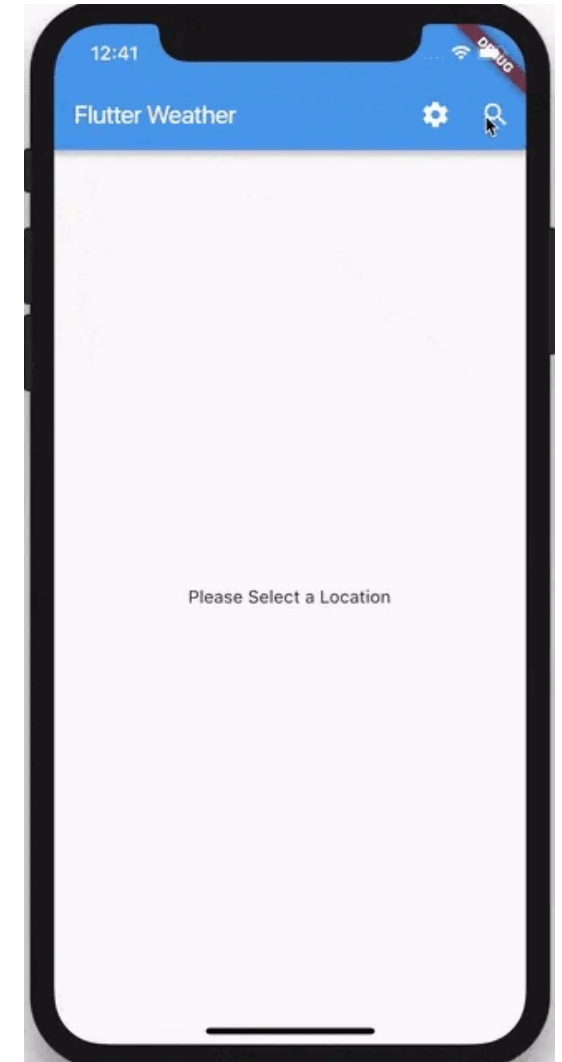
GetX hỗ trợ chuyển đổi ngôn ngữ dễ dàng với `.tr` mà không cần quá nhiều cấu hình.

```
MaterialApp(  
  locale: Locale('en', 'US'),  
  supportedLocales: [  
    Locale('en', 'US'),  
    Locale('es', 'ES'),  
  ],  
  localizationsDelegates: [  
    // các delegate khác  
  ],  
);
```

```
GetMaterialApp(  
  // chỉ cần tạo một lớp với các bản dịch  
  translations: MyTranslations(),  
  locale: Locale('en', 'US'),  
);  
  
Text('hello'.tr);
```

## Xây dựng một ứng dụng thời tiết sử dụng Flutter và Provider/BloC/Get:

1. **Tìm kiếm thành phố:** Người dùng có thể tìm kiếm thông tin thời tiết của một thành phố thông qua trang tìm kiếm.
2. **Hiển thị thông tin thời tiết:** Ứng dụng sẽ hiển thị thời tiết hiện tại của thành phố từ API OpenMeteo.
3. **Thay đổi đơn vị nhiệt độ:** Cho phép người dùng thay đổi giữa các đơn vị nhiệt độ (Celsius và Fahrenheit).
4. **Thay đổi giao diện dựa trên thời tiết:** Giao diện của ứng dụng sẽ thay đổi màu sắc theo tình hình thời tiết của thành phố đã chọn.
5. **Trạng thái ứng dụng:** Sử dụng 1 trong 3 thư viện Provider, BloC hoặc Get



# Cảm ơn

## Các bạn đã chú ý lắng nghe

---



